1. **Поняття БД.**

*Сутності і зв’язки. Модель даних. Реалізація мделі даних. Незалежність даних. Три рівня архітектури БД. Мови даних.*

БД- це представлення в об’єктивній формі сукупності самостійних матеріалів, систематизованих таким чином, щоб ці матеріали могли бути знайдені й оброблені за допомогою електронної обчислювальної машини. Хорошим прикладом БД є телефонний довідник. БД- це поіменована, структурована сукупність логічно взаємопов’язаних даних, які характеризують окрему предметну область і перебувають під СКБД.

Предметна область- частина реального світу, що підлягає вивченню з метою організації управління і в кінцевому рахунку,автоматизації.

Сутність- будь-який конкретний чи абстрактний об’єкт до розглянутої предметної області. Сутності- це базові типи інформації, які зберігається в БД. До сутностей можуть належати студенти, клієнти, підрозділи і т.д.

Атрибут- це властивість сутності в предметній області. Його найменування повинне бути унікальним для конкретного типу сутності. Наприклад, для студента сутності можуть використовувати такі атрибути: прізвище, ім’я, дата і місце народження.

Зв’язок- взаємозв’язок між сутностями в предметній області. Зв’язки представляють собою з’єднання між частинами БД.

Сутності-це дані, які класифікуються за типом, а зв’язки показують, як ці типи даних співвідносяться один і одним.

Модель даних- абстрактне представлення реального світу, що відображає тільки ті об’єкти, що безпосередньо стосуються програми.

Модель даних- це абстрактне, самостійне, логічне визначення об’єктів, операторів та інших елементів, що в сукупності створюють абстрактну машину доступу до даних, з якою працює користувач.

Реалізація моделі даних-це фізичне втілення на реальній машині компонентів абстрактної машини, які в сукупності складають цю модель.

Незалежність даних-властивість, що забезпечує можливість змінювати загальну логічну або фізичну структуру даних без зміни при цьому представлення прикладних програм про дані є одним із принципів побудови банків даних. Стандарт визначає три рівні в архітектурі БД, на кожному з яких описується своя модель, що відображає особливості сприйняття та задач опрацювання даних, характерні для цього рівня:

-зовнішній- передбачає опис підмножин даних із застосуванням понять і термінів, якими оперують користувачі прикладної системи.

-концептуальний- рівень задач розробника та адміністратора БД, на якому описуються системи структури БД.

-внутрішній- рівень системного програмування, на цьому рівні розв’язуються проблеми зберігання даних та доступу до них без специфікації змісту.

1. **Реляційні БД.**

*Три аспекти реляційної моделі даних. Реляційна БД. Операції визначення даних. Операції маніпулювання даними.*

Реляційна модель даних включає такі компоненти: -структурний аспект-дані в БД і набором відносин -аспект цілісності-відносини(таблиці) відповідають певним умовам цілісності -аспект обробки-РМД підтримує оператори маніпулювання відносинами(реляційна алгебра, реляційне числення). Реляційна БД(РБД)-база даних, основана на реляційній моделі даних. Інакше кажучи РБД-це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеня. Кожне подання має підтримувати всі операції маніпулювання даними, які підтримують реляційні таблиці: операції вибірки, вставки, модифікації і видалення даних.

1. **Реляційна модель.**

*Типи даних. Типи, які підтримує SQL. Домени. Кортежі. Відношення.*

Реляцíйна модéль дáних — розроблена Едгаром Коддом в 1970 логічна модель даних, що описує: структури даних у вигляді наборів відношень, що, можливо, змінюються в часі; теоретико-множинні операції над даними: об'єднання, перетин, різниця і декартів добуток; спеціальні реляційні операції: селекція, проекція, з'єднання і розподіл; а також спеціальні правила, що забезпечують цілісність даних. Обробка даних в реляційній моделі ґрунтується на принципах реляційної алгебри. Будь який взятий тип є скалярним або не скалярним. -Нескалярним називають тип, значення якого явно визначені як мають множину визначених користувачеві, безпосередньо доступних користувачеві. -Скалярні називають тип який не є скалярним.Іноді його ще називають капсульованим і атомарним. Типи які підтримує SQL : 1)символьні(nvarchar(n)) 2)бітова стрічка(bit(n),varbit(n)) 3)числові(integer, float, double) 4)збереження цифр(numeric) 5)дата-час(data, time) Домен-обмежена підмножина значень даного типу. Наприклад стать-“Ж” або ”Ч”. Атрибут-стовпчик; кортеж-рядок таблиці(розширений екземпляр).Кожен кортеж є унікальним.Відношення(реляція)-підмножина декартового добутку на якому введено відношення еквівалентності. До одного класу еквівалентності належать тільки ті відношення, які відрізняються порядком компонент в добутку. Відношення- підмножина узагальненого декартового добутку доменів, що зображають властивості всіх примірників сутностей певного класу.

1. **Реляційна алгебра.**

*Операції з відношеннями. Семантика оригінальної алгебри Кодда. Додаткові операції.*

Реляційна алгебра — колекція операцій, що приймають відношення в якості операндів і повертають

відношення в якості результату.

Автор першої версії цієї алгебри - Кодд. Його оригінальна алгебра включала 8 операцій,

які поділяються на 2 групи по 4: 1)Традиційні операцїї з множинами — обєднання, перетин, різниця і декартів добуток.2) Спеціальні реляційні операцїї, такі як скорочення (вибірка), проекція, зєднання і ділення.

***Обєднанням*** двох сумісних по типу відношень A і B називається

відношення з тим же заголовком, що і у відношень A і B, і тілом, що складається з

кортежів, які належать або A, або B, або обом відношенням.

Синтаксис: A UNION B

***Перетином*** двох сумісних по типу відношень A і B називається

відношення з тим же заголовком, що і у відношень A і B, і тілом, що складається з

кортежів, які належать одночасно обом відношенням A і B.

A INTERSECT B

***Різницею*** двох сумісних по типу відношень A і B називається

відношення з тим же заголовком, що і у відношень A і B, і тілом, що складається з

кортежів, які належать A і не належать B.

A MINUS B

***Декартовим добутком*** двох відношень A(A1,A2,…,An) i B(B1,B2,…,Bm)

називається відношення, заголовок якого є ***зчепленням***

***заголовків*** відношень A і B :

(A1,…,An, B1,…,Bm)

а тіло складається з кортежів, які є ***зчепленням кортежів*** відношень A і B:

(a1…an,b1…bn)

таких, що(a1…an) Є A, (b1…bn) Є B.

A TIMES B

***Вибіркою (обмеженням, селекцією)*** на відношенні A з умовою C

називається відношення з тим же заголовком, що і у відношення A, , і тілом, що складається з кортежів, значення атрибутів яких при підстановці в умову C дають значення ІСТИНА. C - логічний вираз, в який можуть входити

атрибути відношення A і (або) скалярні вирази.

A WHERE C

***Проекцією*** відношення A по атрибутах X,Y,…,Z, де кожен з

атрибутів належить відношенню A, називається відношення з заголовком(X,Y,…,Z) і

тілом, що містить множину кортежів виду(x,y,…,z), таких, для яких у відношенні A

знайдуться кортежі зі значенням атрибута X рівним x, значенням атрибута Y рівним y

, …, значенням атрибута Z рівним z.

***Зєднанням*** відношень A і B по умові C називається відношення

(A TIMES B)WHERE C

C - логічний вираз, в який можуть входити

атрибути відношень A і B і (або) скалярні вирази.

Нехай дано відношення A(X1,X2,…,Xn, Y1,Y2,…,Ym) і B(Y1,Y2,…,Ym),

причому атрибути (Y1,Y2,…,Ym) - спільні для двох відношень. ***Діленням*** відношень ***A*** на B

називається відношення з заголовком і тілом, що містить множину кортежів (x1,x2,…,xn) таких, що для *всіх* кортежів(y1,y2,…,ym) у відношенні A

знайдеться кортеж(x1,x2,…,xn, y1,y2,…,ym).

1. **Цілісність даних.**

*Обмеження цілісності. Ключі. Тригери.*

Цілісність даних – система правил для підтримки зв’язків між записами у зв’язаних таблицях, а також для забезпечення захисту від випадкового вилучення або зміни зв’язків даних. Обмеження типу-визначення множин і значень , з яких складається певний тип. Обмеження атрибута- обмеження за значенням, які можна приймати атрибуту.

Обмеження змінної відношення- обмеження за значенням, які дозволено приймати вказаній БД. Ключі у відношеннях як поняття ґрунтується на понятті функціональної залежності і забезпечує можливість застосування в процесах побудови та застосування баз даних певних методів та засобів, недопустимих без застосування ключів.Ключі утворюють мінімальний набір атрибутів які однозначно вказують на всі інші атрибути відношення.

Зовнішній ключ- множина атрибутів деякого відношення R2 значення яких повинно співпадати зі значенням деякого потенційного ключа відношення R1.

Foreing key {< >}REFERANCES < >

Потенційний ключ клужить для того щоб і відношенні R в будь який момент ніякі два кортежі не були дублікатами один одного. PRYMERY KEY< > .

Трегер –це збережена процедура особливого типу, яку користувач не викликає особисто, а використання якої обумовлено настаннями визначеної події {INSERT, DLETE, UPDATE}.Момент запуску починається за допомогою ключових слів BEFORE(до події пр.. дод. даних) або AFTER(після)

1. **Проектування баз даних.**

*Функціональні залежності. Їх властивості. Нескоротність залежностей..*

Нехай r - відношення, а X і Y — певні підмножини множини атрибутів відношення r. Тоді Y

функціонально залежнавід X, що можна записати так:

X → Y(читається "X функціонально визначаєY") тоді і тільки тоді, коли кожне значення множини Х відношення r повязано точно з одним значенням множини Y відношення r.Інакше кажучи, якщо два кортежі відношення r співпадають по значенню Х, вони співпадають і по значенню Y. Це зв'язок типу "багато до одного” між двома множинами атрибутів заданої змінної відношення (це найрозповсюдженіший і важливий вид обмеження цілісності).

Властивості:

1. Правило рефлексивності**.** Якщо множина В є підмножиною множини А,

то А → В.

2. Правило доповнення. Якщо А → B, то АС → ВС.

3. Правило транзитивності. Якщо А → B и B→C, то А → С.

4. Правило самовизначення. А → А.

5. Правило декомпозицїї. Якщо А → ВС, то А → Bи A → C.

6. Правило обєднання. Якщо А → В И А → С, то А → ВС.

7. Правило композицїї. Якщо А → B и С → D, то АС → BD.

8.Правило Дарвена. Якщо А→ B і C → D, то А ∪ ( С - В ) → BD (" ∪ "

операція обєднання множин, а "-" — операція різниці.

Множина функціональних залежностей назив. нескоротною  **тоді і тільки тоді коли коли вона** володіє всіма трьома властивостями свойствами.

1. Права (залежна) частина кожної функціональної залежності з множини S містить тільки один атрибут (є одноэлементною множиною).

2. Ліва частина (детермінант) кожної функціональної залежності з множини S, є нескоротною , тобто жоден атрибут з детермінанту не може бути опущеним без зміни замикання S+ (без перетворення множини S вякусь іншу множину, не еквівалентну множині S). В цьому випадку функціональна залежність називається нескоротною.

3. Жодна функціональна залежність з множини S не може бути видалена з множини S без зміни її замикання S+ (без перетворення множини S вякусь іншу множину, не еквівалентну множині S).

1. **1-2 нормальні форми 457**

*Декомпозиція. Теорема Хіта. Перша нормальна форма. Друга нормальна форма.*

Процедура нормалізації передбачає розбиття, тобто декомпозицію, даної змінної відношення на інші змінні відношення, причому декомпозиція повинна буде оборотною, тобто виконуватись без втрат інформації.

Теорема Хіта. Нехай R(A, B, C) є відношенням, де А, В, С – множини атрибутів цього відношення. Якщо В функціонально залежить від А (А→В), то R дорівнює поєднанню її проекцій по атрибутах { A, B }і{А, С}.

Перша нормальна форма. Будь-яке відношення, записане у табличній формі, є у 1НФ. Відношення мають такі властивості: 1.всі значення атрибутів атомарні. 2.атрибути не впорядковані і різняться за назвою. 3. у відношенні не має однакових кортежів. 4.кортежі не впорядковані.

Потенційний ключ – атрибут або набір атрибутів, які однозначно визначають кортеж.

Первинний ключ – один із потенційних ключів, який вважається головним. Він має складатися із мінімальної кількості атрибутів. Ключ, що містить 1 атрибут, назив. простим, складається із 2х і більше – складеним.

Функціональна залежність: нехай R – відношення. Множина атрибутів Y функціонально залежить від множини атрибутів X (X функціонально визначає Y) тоді і тільки тоді, коли для будь-якого стану відношення R для будь-яких кортежів r1, r2 з R з того, що r1.X=r2.X випливає r1.Y=r2.Y. Записують це так: X→Y. X називають детермінантом, а Y – залежною частиною.

Друга нормальна форма. Якщо відношення є в 1НФ і нема функціональних залежностей між неключовими атрибутами та частиною складеного ключа. Якщо є функціональна залежність між неключовими атрибутами та частиною складеного ключа, то будуємо нове відношення, в яке виносимо детермінант в якості первинного ключа та праву частину залежності.

1. **3 нормальна форма. Нормальна форма Бойса-Кодда.**

*Незалежні проекції. Збереження залежностей. Тривіальна функціональна залежність.*

Атрибути називаються взаємно незалежними, якщо жоден із них не є функціонально залежним один від одного.

3НФ. Якщо відношення є у 2НФ і нема взаємно незалежних неключових атрибутів, то воно є у 3НФ.

НФБК. Відношення є у НФБК, якщо детермінанти всіх функціональних залежностей є первинними ключами.

ФЗ називається тривіальною, якщо залежна частина є підмножиною детермінанта.

1. **Нормальні форми вищих порядків.**

*Багатозначні функціональні залежності. Четверта нормальна форма. Залежність сполучення. Пята нормальна форма.*

Нормалізація починаючи від НФБК пов’язана з поняттям багатофункціональної залежності. Атрибути (множина атрибутів) Y i Z багатозначно залежать від Х, тоді і тільки тоді, коли з того, що відношення R яке має кортежі r1=(x,y,z1) і r2=(x,y1,z) випливає, що в відношенні R є також кортеж r3=(x,y,z). Коректність подальшої декомпозиції лежить на теоремі Фейджина. Якщо у відношенні є функціональна залежність , то автоматично є і тривіальна багатозначна залежність, яка визначається цією функціональною залежністю. Багатозначна залежність X🡪Y|Z називається нетривіальною багатозначною залежністю, якщо не існує функціональної залежності X🡪Y, X🡪Z. Відношення R знаходиться в 4НФ тоді і тільки тоді, коли відношення знаходиться в НФБК і не містить нетривіальних багатозначних залежностей. Мають місце залежності спеціального вигляду, коли відношення не може бути декомпозиційоване без втрати на дві проекції, але може бути декомпозиційоване на більше число проекцій. Тікі залежості називаються залежностями зєдняння. Відношення знаходиться в 5НФ тоді і тільки тоді, коли будь-яка залежність зєдняння є тривіальною.

1. **Графічне представлення реляційної моделі.**

*ER-діаграми. UML-діаграми для моделювання даних.*

Сутність-це клас однотипних об’єктів, інформація про які має бути описана в модлі. Має містити назву, виражене єдиним числом. Екземпляр сутності-це характеристика, властивість сутності. Ключ сутності- це набір атрибутів, значення яких є унікальним для кожного екземпляра сутності. При розробці ER-діаграми маємо мати інфу про предметну область. А)Список сутностей. Б)список атрибутів сутностей. В) Взаємозвязки між сутностями.

Розрізняють концептуальні і фізичні ER-діаграми. Концептуальні не враховують особливості конкретних СУБД. Фізичні будуються по концептуальних і являють собою прообраз конкретної БД. В діаграмі сутності є таблицями, атрибути-колонками таблиць, зв’язки реалізуються міграцією ключових атрибутів батьківських сутностей і утворення зовнішніх ключів. При правильному визначенні сутностей отримані таблиці зразу знаходяться в 3НФ. UML є мовою широкого профілю, це [відкритий стандарт](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B8%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), що використовує графічні позначення для створення [абстрактної моделі](http://uk.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) [системи](http://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), називаної *UML-моделлю*. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація. Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

1. **Індексування та методи доступу до даних.**

*Індекси для впорядкованих даних. Стратегії керування індексами. Вторинні індекси. Індекси на основі B-дерев та геш-таблиць.*

Ознайомимось з індексами з найпростішого варіанта структури: посортованому *файлові даних* (*data file)* відповідає *файл індексу (index file)*, сформований з пар виду *«ключ-покажчик»*. Значення *К* ключа пошуку в файлі індексу асоційоване з покажчиком, який посилається на запис файла даних, що володіє значенням *К*. Подібна індексна структура може бути *щільною (dense)* у тому розумінні, що *кожному* запису файла даних відповідає певний елемент файла індексу, або *розрідженою (sparse)* – файл індексу містить покажчики тільки на деякі записи файла даних (наприклад, один елемент з розрахунку на кожен блок даних). На основі посортованих записів файла даних можна створити *щільний індекс*, який являє собою послідовність блоків, що містять ключі записів даних і покажчики на ці записи (під покажчиками розуміють адреси – у тому розумінні, як говорилось у темі 10). **Щільним називають індекс, який містить ключі для *кожного* запису файла даних.** Якщо щільний індекс виявиться надто великим, то інколи доцільно скористатися подібною структурою *розрідженого індексу*, яка дає змогу зменшити розмір файла ціною ймовірного збільшення проміжку часу, необхідного для відшукання запису за заданим значенням ключа.

Іноді виникає необхідність створення *декількох* індексів для одного і того ж відношення, які дають змогу пришвидшити обробку запитів різних категорій .Вторинний індекс – це щільний індекс, який, зазвичай, містить дублікати ключових значень. Як і раніше, індекс складається з пар значень виду *«ключ-покажчик»*. «Ключ» у цьому випадку – це ключ пошуку, і вимоги щодо його унікальності не ставлять. Елементи файла індексу посортовані за значеннями ключа – це дає змогу пришвидшити пошук необхідного елемента. Вторинні індекси часто використовують і для оптимізації доступу до структур іншої категорії, які часто використовуються і називаються *компактно згрупованими файлами (clustered files).*

***В*-дерева**

її блоки організовані у вигляді *деревоподібного графа* (*tree-like graph*). *В*-дерево є *збалансованим* – у тому розумінні, що довжини всіх шляхів від *кореневої* (*root*) вершини до будь-якої з вершин-*листів* (*leaves*) є рівними. Типове *В*-дерево містить три рівні: кореневу вершину, проміжні вершини і листи. Кожному *В*-деревоподібному індексу відповідає параметр *n*, який визначає властивості компонування блоків *В*-дерева. Кожен блок має простір, достатній для розміщення *n* значень ключа пошуку і *n+1* покажчиків. Блок дерева насправді подібний до індексних блоків, які розглянуто у темі 13, за винятком того, що поряд з *n* парами виду *«ключ-покажчик»* він містить додатковий, *n+1*-й, покажчик. Величина *n* обирається так, щоб забезпечити можливість зберігання у блоці *n+1* покажчиків і *n* ключових значень. Модель *В*-дерева є потужним інструментом конструювання індексів. Послідовність покажчиків на записи, які містяться у вершинах-листах, здатна виконувати ті ж функції, що і будь-яка підмножина покажчиків у «традиційних» індексних файлах.

Існує чимало корисних структур індексів, які реалізують модель *геш-таблиць* (*hash tables*). Кожній подібній структурі відповідає деяка *геш-функція* (*hash function*), яка отримує як параметр значення ключа пошуку (у даному випадку варто називати його *геш-ключем* − *hash key*) і обчислює число з інтервалу від *0* до *В-1*, де *В* – кількість *сегментів* (*buckets*). Елементи *масиву сегментів* (*bucket array*) проіндексовані від *0* до *В-1* і містять заголовки зв’язаних списків (їхня кількість дорівнює *В*), по одному на кожен елемент-сегмент масиву. Якщо запис володіє значенням *К* ключа пошуку, то він приєднується до списку сегмента з номером *h(K)*, де *h* – геш-функція.

1. **Багатовимірні індекси**

*Сітки. Розділювані геш-функції. KD-дерева. Quad-дерева. R-дерева.*

Перша структура, (сітковий файл – grid file), зазвичай, не передбачає «перемішування» (hash) значень за вимірами, однак розділяє виміри шляхом сортування значень, які до них належать.

*Сіткові файли*

Одну з найпростіших схем індексації, яка часто перевищує за ефективністю традиційні індекси, що використовуються для обробки запитів до багатовимірних даних, називають сітковим файлом. Модель сіткового файла можна представити так, ніби простір точок-об’єктів розбивається на частини уявною сіткою. Лінії сітки, паралельні до осей кожного виміру, ділять простір на смуги. Точки, що належать лінії, яка є нижньою межею певної смуги, належать до цієї смуги. Кількість ліній сітки за різними вимірами може бути різною. Окрім того, ширина смуг також може бути різною − навіть у межах одного виміру.

*Роздільні геш-функції*

Геш-функції як аргумент здатні отримувати списки значень атрибутів, хоча, зазвичай, гешуванню піддаються значення одного атрибута. Наприклад, якщо a і b – цілочисловий і рядковий атрибути, відповідно, то для обчислення номера сегмента геш-таблиці, використаної як індекс для пар значень (а,b), можна додати величину а з ASCII-кодами усіх символів рядка b, поділити отриману суму на кількість сегментів і взяти залишок від ділення. Однак подібну геш-функцію можна використати тільки в таких запитах, в яких водночас беруть участь значення атрибутів а і b. Краще ж проектувати геш-функцію так, щоб вона повертала певну кількість бітів (наприклад, k), розподілених між n атрибутами, де ki бітів геш-коду обчислюють на підставі і-го атрибута і . Якщо казати точніше, то геш-функція h насправді є списком (h1. h2. …, hn) геш-функцій, де функція hi застосовується до значення і-го атрибута і повертає послідовність з ki бітів. Номер сегмента індексу, з якого адресується кортеж зі значеннями (v1, v2, …, vn) n атрибутів, обчислюється шляхом зчеплення часткових бітових послідовностей: h1(v1)h2(v2)…hn(vn). kknii=Σ=1

*Використання kd-дерев для побудови індексів*

Структуру, яка слугує для представлення інформації в ОП й узагальнює модель бінарних дерев пошуку на випадок багатовимірних даних, називають kd-деревом (k-dimensional search tree). Адже kd-дерево – це бінарне дерево, з кореневою і проміжними вершинами якого асоційовані атрибут а, що представляє деяку розмірність даних, і певне значення v цього атрибута, яке поділяє множину точок даних на дві підмножини: точкам однієї підмножини відповідають значення атрибута а, менші від v, а точкам другої – значення а, рівні або більші від v. Атрибути в межах одного рівня дерева однакові, а в сусідніх рівнях – різні: рухаючись від кореневої вершини дерева «вниз», рівнями проміжних вершин, усі атрибути розмірності циклічно заміщають один одного.

*Дерева квадрантів*

У дереві квадрантів (quad tree) кожна проміжна вершина відповідає певному квадранту простору даних – прямокутній двовимірній області або k-вимірному паралелепіпеду, якщо кількість вимірів дорівнює k. Як і під час розгляду решти структур даних, що описуються в цьому розділі, ми зосередимо увагу на випадку двох вимірів. Якщо кількість точок, які належать квадранту, нижча від припустимого максимуму, то квадрант треба сприймати як вершину-лист дерева, що представляється

відповідним блоком. Якщо ж, навпаки, кількість точок надто велика, то квадрант інтерпретується як проміжна (на початку – коренева) вершина з чотирма дочірніми вершинами, що відповідають частковим квадрантам, застосовні і до моделі дерев квадрантів.

*R-дерева*

R-дерево(англ. R-trees)- деревоподібна структура даних(дерево),подібно B-дереву, але використовується для організації доступу до просторових даних, тобто для індексації багатовимірної інформації,такої,наприклад, як географічні дані з двовимірними координатами (широтою і довготою). Ця структура даних розбиває простір на безліч ієрархічно вкладених і, можливо,що перетинаються,

прямокутників (для двовимірного простору). У разі тривимірного або багатовимірного простору це будуть прямокутні паралелепіпеди .

1. **Оптимізація схеми бази даних.**

*Денормалізація. Ортогональне проектування. Перетворення виразів. Оптимізатор запитів.*

Нехай R1,R2,..,RN- множина змінних відношення. Тоді денормалізацією є заміна їх з’єднанням R, що для всіх і=1,.., N виконання проекції R за атрибутами Rі знову приведе до створення значення R. Денормалізація приводить до надлишковості даних, але інколи після її виконання запити виконуються швидше. Principle of orthogonal design: Нехай А і В – 2 різні базові змінні відношення в деякій бд. Тоді для А і В повинна існувати декомпозиція без втрат на такі проекції А1,..Аm і В1,..Вn, що довільні Аі та Вк будуть володіт смисловими значеннями, що не перекриваються. Нормалізація скорочує надлишковість даних всередині змінних відношення, а ортогональне проектування скорочує надлишковість даних між змінними відношення. Оптимізатор володіє статистичними даними про бд.

Процес оптимізації:

1. Перетворення запиту у внутрішню форму(побудова абстрактного синтаксичного дерева – дерева запиту)
2. Перетворення запиту у канонічну форму
3. Вибір потенційних низькорівневих процедур(вибір шляху доступу)
4. Генерація різних варіантів планів обчислення запиту і вибір плану з мінімальними затратами.

Перетворення виразів:

1. Операції скорочення і проекції

а) послідовність операцій скорочення одного й того ж відношення м. б. замінена на єдину операцію скорочення цього відношення(умови об’єднуються через and)

б) у послідовності операцій проекції для одного й того ж відношення можна ігнорувати всі проекції, окрім останньої

в) скорочення для результату проекції модна перетворити у операції проекції для результату скорочення

Скорочення бажано робити перед проекцією

1. Розподільчий закон:

Унарний оператор f розподіляється по бінарній операції wтоді і лише тоді, якщо для довільних A та B : f(AwB)=f(A)wf(B). Операція скорочення розподіляється по операціях об’єднання, перетину, різниці, інколи – сполучення. Операція проекції розподіляється по об’єднанню, перетину, інколи – сполученню.

1. Комутативність(об’єднання, перетину, сполучення) і асоціативність(об’єднання, перетину, сполучення)
2. Ідемпотентність(об’єднання, перетину, сполучення) і поглинання (A UNION (A INTERSECT B)=A, A INTERSECT (A UNION B)=A)
3. Обчислювані вирази (Бінарна операція f розподіляється по бінарній операції wтоді і лише тоді, якщо для довільних A та B та С: Аf(AwB)=(AfВ)w(АfC).)
4. Логічні вирази (перетворюються на основі транзитивності операцій та твердження про те, що довільний логічний вираз може бути перетворений у еквівалентний у КНФ . Система логічно виводить інформацію про наперед застосовні операції скорочення)
5. Семантичні перетворення(допустимі в силу того, що існує конкретне установлене обмеження цілісності)
6. **Управління доступом в базах даних.**

*Поняття аутентикації та авторизації. Підходи до організації захисту даних.*

Поняття аутентикації та авторизації. Підходи до організації захисту даних.

Ідентифікація- призначення користувачу(об’єкту-споживачу ресурсів сервера) унікального імені.

Аутентифікація – підтвердження істинності(дійсності) користувача, що надав ідентифікатор.

Авторизація – визначення переліку конкретних інформаційних ресурсів та операцій з ними, з якими аутентифікованому користувачу дозволено працювати.

Аутентифікація може бути як і внутрішньою(засобами СУБД), так і зовнішньою(засобами ОС, мережі, біометрична). Процес доступу до інформації заснований на попередньому визначенні суб’єкта, що запитує доступ, і об’єкта доступу.

Ідентифікація починається зі встановлення з'єднання користувача зі системою. Він надає системі певні параметри, якщо підсистема аутентифікації їх приймає,то встановлюється сесія взаємодії користувача з бд. Процедура аутентифікації з математичної точки зору є перевіркою статистичної гіпотези, тобто можливими є помилки 1 та 2 роду. Ця процедура заснована на знанні(пароль, логін). Логін(унікальний ідентифікатор) є заявкою на ідентифікацію, а пароль (парольна фраза)– підтвердженням цієї заявки. Під захистом даних будемо розуміти попередження доступу до них зі сторони несанкціонованих користувачів. У СУБД підтримується один із 2методів організації захисту даних(інколи і обидва):

1. Вибірковий(discretionary)Кожному користувачу призначаються різні права доступу

(привілеї) для різних об’єктів. Як правило різні користувачі володіють різними правами доступу до одного і того ж об’єкта. Тому такий метод характеризується значною гнучкістю. (Все, що явно недозволено, неявно заборонено). В деяких випадках ведеться контрольний журнал(audit trail) , до якого вноситься інформація про всі події, що відбуваються у системі.

1. Мандаторний(mandatory) Кожному об’єкту присвоюється певний класифікаційний рівень(classification level), а кожному користувачу присвоюється певний рівень доступу(clearance level), з градаціями, що відповідають існуючим класифікаційним рівням. Передбачаються, що ці рівні утворюють строгу ієрархічну систему. В результаті право доступу до об’єкта мають тільки користувачі з відповідним рівнем доступу. Два правила доступу: а) Користуач може здійснити вибірку даних об’єкта к тільки тоді, якщо його рівень доступу більший рівний класифікаційного рівня доступу об’єкта(проста властивість безпеки – simple security property). Б) Користувач к може модифікувати об’єкт і тільки якщо його рівень доступу рівний класифікаційному рівню об’єкта і(зіркове правило – star property).

Об’єктом даних , для якого організовується захист, може бути як і окремий кортеж(чи його компонент), так і вся база даних.

**15.Обмеження змінної відношення і бази даних.**

Обмеження змінної відношення полягає в тому шо обмеження переходу здійснюється тільки на одну змінну відношення, а бази даних на дві і більше змінних відношення. Обмеження переходу – це обмеження, яке встановлює правила переходу певної змінної. Нарпиклад: поле в якому заповнюється «ніколи не був одружений» може мінятися на «одружений» , але не може бути змінене навпаки.

*Тригери*

Тригери – це процедури, які зберігаються разом з базою даних, і викликаються тоді коли виконуються певні умови.

Зауваження:

1. Тригери не є рекомендованими засобами обмеження. Викликають труднощі в оптимізації бази даних.
2. Зазвичай тригери використовують в обмеженні, однак вони можуть використовуватися в: передача користувачу попереджувального повідомлення і тд.
3. Тригер може бути процедурою виконання різноманітних дій за певних умов.
4. Транзакції та блокування.
5. **Транзакції та блокування**

Відновлення в системі баз даних означає, в першу чергу, відновлення самої бази даних, тобто повернення бази даних в певний стан, який вважається правильним, після деякого збою, в результаті якого поточний стан стає неправильним або, по крайній мірі,  
досить невизначеним.

*Відновлення системи.*

Система повинна бути готова до відновлення не тільки після локальних відмов,а й після глобальних порушень, наприклад відключення електроенергії. Локальне порушення впливає тільки на ту транзакцію, в якій воно, власне кажучи, і сталося. Глобальне порушення впливає відразу на всі транзакції, які виконувалися в момент його виникнення, і тому призводить до більш значних для системи наслідків.

Критичним моментом у відмові системи є втрата вмісту основної  
(оперативної) пам'яті (зокрема, буферів бази даних). Оскільки точний стан  
будь-якої транзакції, що виконувалася в момент відмови системи залишається невідомим,така транзакція не може бути успішно завершена. Тому при перезапуску системи  
будь-яка така транзакція буде скасована.

Виникає очевидне питання: як система визначає в процесі перезапуску, яку  
транзакцію слід скасувати, а яку виконати повторно? Відповідь полягає в тому,  
що система автоматично створює контрольні точки з деяким наперед заданим  
інтервалом. Для створення контрольної точки потрібно, по-перше, виконати примусове збереження вмісту буферів оперативної пам'яті у фізичній базі даних, і, по-друге, здійснити примусове збереження спеціального запису контрольної точки в журналі на фізичному носії. Запис контрольної точки містить  список всіх транзакцій, які виконуються в той момент, коли створювалася контрольна точка.

*ACID-властивості транзакцій.*

ACID - це скорочене позначення таких властивостей транзакцій, як нерозривність, правильність, ізольованість і стійкість (atomicity- correctness-isolation-durability).

Нерозривність. Будь-яка конкретна транзакція діє за принципом "все або  
нічого ".

Правильність. Будь-яка конкретна транзакція перетворює правильний стан бази даних в  
інший правильний стан, при тому що збереження властивості правильності на  
всіх проміжних етапах цього перетворення не є обов'язковим.

Ізольованість. Будь-які оновлення, внесені конкретною транзакцією, залишаються прихованими від усіх інших транзакцій до тих пір, поки не відбудеться фіксація даної транзакції.

Стійкість. Після фіксації будь-якої конкретної транзакції внесені нею обновлення залишаються в базі даних, навіть якщо згодом відбудеться аварійна зупинка системи.

**17. Паралельна обробка даних.**

*Типи блокувань. Рівні ізоляції транзакцій. Блокування наміру.*

Термін паралельність позначає таку властивість СУБД, яка, як правило, дозволяє одночасно звертатися за допомогою багатьох транзакцій до однієї і тієї ж бази даних. Очевидно, що в системі, яка має таку властивість, потрібен певний механізм управління, який дозволяє домогтися того, щоб паралельно виконувані транзакції не порушували роботу один одного.

*Типи блокувань.*

Всі проблеми організації паралельної роботи, можуть бути  
усунені за допомогою механізму управління паралельним виконанням, який називають  
блокуванням. В його основі лежить проста ідея - якщо для деякої транзакції А потрібна гарантія, щоб певний об'єкт, в якому вона зацікавлена, не змінився якимось чином без її відома, вона набуває блокування на цей об'єкт.

*Рівні ізоляції транзакцій.*

Упорядкованість  гарантує ізольованість транзакцій, в тому трактуванні цього  
терміна, яке  застосовується при описі властивостей ACID. Одним з безпосередніх і досить сприятливих наслідків з цього факту є те, що якщо всі графіки  
- впорядковувані, то прикладний програміст, розробляючи код  для будь-якої   
конкретної  транзакції А, не повинен звертати абсолютно ніякої уваги на той факт,  
що одночасно з нею в системі може виконуватися деяка інша транзакція В.  
Але можна також стверджувати, що протоколи, що застосовуються для забезпечення упорядкованості, знижують ступінь розпаралелювання, або загальну продуктивність системи, до неприйнятних рівнів. Тому на практиці в системах звичайно підтримується цілий ряд інших рівнів "ізоляції" .

Може бути визначено щонайменше п'ять різних рівнів ізоляції, але в стандарті SQL і в СУБД DB2 підтримуються лише чотири. Взагалі кажучи, чим вище  
рівень ізоляції, тим менший ступінь втручання транзакцій в роботу один одного (і  
тим нижчий ступінь розпаралелювання), а чим нижче рівень ізоляції, тим більше ступінь  
втручання (і вище ступінь розпаралелювання).

**Блокування наміру.**

протокол блокування наміру-це протокол згідно з яким жодній транзакції не дозволяється отримати блокування на кортежі перед тим, як буде спочатку отримане блокування на змінну відношення,  
яка її містить.

Єтри типу блокування наміру:роздільні (Intent Shared - IS) блокування, виняткові (Intent Exclusive - IX) блокування і роздільні виняткові (Shared Intent Exclusive - SIX)  
блокування.

**18. Проблема взаємного блокування.**

*Дедлок. Необхідні умови. Рівні ізоляції транзацій. Версійна модель транзакцій.*

*Взає́мне блокува́ння* — ситуація, коли кожен із групи [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) очікує на подію, яку може викликати лише інший процес з цієї групи.. Також зустрічаються назви *тупикова ситуація*, *тупик*, [*клінч*](http://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D0%BD%D1%87). В англомовній літературі ця ситуація має назву [англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0)***deadlock*** (вимовляється як *дедлок*).В галузі [інформаційних технологій](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D1%96_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%97), взаємним блокуванням називають ситуацію, коли два або більше [процесів](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81) чекають поки інший не не звільнить певний ресурс, або коли більше ніж два процеси чекають на звільнення ресурсів в замкненому ланцюгу.

*Рівні ізоляції транзакції*

*Незавершенное чтение (read uncommitted)*

Минимальный уровень изоляции, гарантирует  только физическую целостность при записи данных. Процессы-читатели могут считывать данные незавершенной транзакции процесса-писателя.

*Подтвержденное чтение (read committed)*

Процессы-читатели не могут считывать данные незавершенной транзакции, но процессы-писатели могут изменять уже прочитанные читателем данные.

*Повторяемое чтение (repeatable read)*

Повторное чтение данных вернет те же значения, что и в начале транзакции. При этом процессы-писатели могут вставлять новые записи, имеющие статус "фантома" при незавершенной транзакции.

*Версионный срез (snapshot)*

Процессы-читатели не ждут завершения транзакций писателей, а считывают данные, точнее их версию, по состоянию на момент начала своей транзакции.

*Сериализуемость (serializable)*

Максимальный уровень изоляции, гарантирует неизменяемость данных другими процессами до завершения транзакции.

*Версійна модель транзакцій*

Процессы-читатели не ждут завершения транзакций писателей, а считывают данные, точнее их версию, по состоянию на момент начала своей транзакции.

**19.Нереаляційні моделі даних.ієрархічна модель**  
*Ієрархічна модель. Сіткова модель.*  
Ієрархічна модель даних організовує дані в виді деревовидної структури.Це є ієрархія батьківських і дочірних сегментів даних.  Ця структура припускає, що записи можуть мати повторення інформації в дочірних сегментах  даних. Дані є серією записів, які мають множину  значень полів . Він збирає всі випадки специфічних  записи разом, як тип запису. Ці типи записів є еквівалентом таблиці в реляційної моделі, так і з окремими записами є еквівалентом рядка. Для створення зв'язку між цими типами записів, ієрархічна модель використовує <<батьківсько-дочірний>> звязок. Це 1:N відображення між типами записів. Це робиться за допомогою дерев.

Наприклад, організація може зберігати інформацію про співробітника, таку як ім'я, номер співробітника, відділ, заробітну плату. Організація може також зберігати інформацію про дітей співробітників, таку як ім'я і дату народження. Працівник і діти  формують ієрархію, де дані про співробітників представляє батьківський сегмент і дані про дітей представлені  в дочірному сегменті. Якщо працівник має трьох дітей, то буде три дочірні сегменти пов'язані з одним співробітником сегменті.

В ієрархічній базі даних батько-нащадок є один до багатьох. Це обмежує дочірний сегмент мати мати тільки одного батьківського сегмента. Ієрархічні СУБД були популярні в кінці 1960-х.

*Сіткова модель*

У мережній моделі даних поняття головних і підлеглих об'єктів дещо розширені. Будь який об'єкт може бути і головним, і підлеглим (у мережній моделі головний об'єкт позначається терміном «власник набору», а підлеглий — терміном «член набору»). Той самий об'єкт може одночасно виконувати і роль власника, і роль члена набору. Це означає, що кожний об'єкт може брати участь у будь-якій кількості взаємозв'язків.  
Подібно до ієрархічної, мережну модель також можна подати у вигляді орієнтованого графа. Але в цьому випадку граф може містити цикли, тобто вершина може мати кілька батьківських вершин.  
Така структура набагато гнучкіша і виразніша від попередньої і придатна для моделювання більш ширшого класу завдань.

20. **Напівструктурована модель даних.(неповністю)**

*XML. Коректність. Валідність. Способи опису структури XML. Перетворення XML документів.*

Моделі напівструктурованих даних в системах баз даних відводиться важлива роль:

1. Модель виконує функцію зручного засобу інтеграції даних, тобто опису з допомогою одних і тих же термінів подібної інформації, яка міститься в декількох базах даних, забезпечених різними схемами.
2. Модель корисна для представлення документів в інструментальних системах, подібних XML, які використовуються для забезпечення доступу до даних в середовищі WEB.

XML. eXtensible Markup Language – це стандарт, розроблений WWW Consortium і реалізуючий модель напівструктурованих даних в застосуванні до задачі опису текстових документів. Вершини графа напівструктурованих даних відповідають в документі елементам тексту, а деякі позначені дуги представляються парами початкових і кінцевих тегів. Теги – стрічки тексту, поміщені в кутові скоби <>.

XML спроектована як використання одного з двох – режимів.

1. Коректний(правильний-well-formed) документ XML дозволяє автору створювати власні теги, подібні міткам дуг в графі напівструктурованих даних. Цей режим у великій мірі відповідає моделі напівструктурованих даних, в якій попереднє визначення яких небуть схем не потребує: в документі можуть міститися будь які теги, включені за визначенням автора.
2. Дійсний(valid) документ XML передбачає наявність DTD; DTD регламентує номенклатуру доступних до застосування тегів і правила їх взаємного вложення і інтерпретації. Категорію дійсних документів XML можна вважати деяким компромісом між такими моделями, такими як реляційна або ODL, які припускають задання жорстких схем і моделлю напівструктурованих даних, в якій схеми як такі зовсім не застосовуються.

**21. Об’єктно-орієнтовна модель**

*Об'єктно-реляційні відображення (ORM). Особливості. переваги і недоліки*

ORM ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Object-relational mapping*, Обє'ктно-реляційна проекція) — технологія [програмування](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), яка зв'язує [бази даних](http://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) з концепціями [об'єктно-орієнтованих](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) мов програмування, створюючи «віртуальну об'єктну базу даних».

Проблема

У об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна розглянути адресну книгу, яка містить список людей разом з кількома телефонами і кількома адресами. В термінах об'єктно-орієнтованого програмування вони представлятимуться об'єктами класу «Чоловік», які міститимуть наступний список полів: ім'я, список (або масив) телефонів і список адрес.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відносин між ними. Ці об'єкти називають «постійними» ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *persistent*). Історично існує кілька підходів до рішення цієї задачі.

Реляційні СУБД

Вирішення проблеми зберігання даних існує — це реляційні системи управління базами даних. Використання реляційної бази даних для зберігання об'єктно-орієнтованих даних приводить до семантичного провалу, примушуючи програмістів писати програмне забезпечення, яке повинне уміти як обробляти дані в об'єктно-орієнтованому вигляді, так і вміти зберегти ці дані в реляційній формі. Ця постійна необхідність в перетворенні між двома різними формами даних не тільки сильно знижує продуктивність, але і створює труднощі для програмістів, оскільки обидві форми даних накладають обмеження одна на одну.

Реляційні бази даних використовують набір таблиць, що представляють прості дані. Додаткова або зв'язана інформація зберігається в інших таблицях. Часто для зберігання одного об'єкта в реляційній базі даних використовується декілька таблиць; це, у свою чергу, вимагає застосування операції JOIN для отримання всієї інформації, що відноситься до об'єкта, для її обробки. Наприклад, в розглянутому варіанті із записною книгою, для зберігання даних, швидше за все, використовуватимуться як мінімум дві таблиці: люди і адреси, і, можливо, навіть таблиця з телефонними номерами.

Оскільки системи управління реляційними базами даних зазвичай не реалізують реляційного представлення фізичного рівня зв'язків, виконання кількох послідовних запитів (що відносяться до однієї «об'єктно-орієнтованої» структури даних) може бути дуже витратне. Зокрема, один запит вигляду «знайти такого-то користувача і всі його телефони і всі його адреси і повернути їх у такому форматі», швидше за все, буде виконаний швидше за серію запитів вигляду «Знайти користувача. Знайти його адреси. Знайти його телефони». Це відбувається завдяки роботі оптимізатора і витратам на синтаксичний аналіз запиту.

Деякі реалізації ORM автоматично синхронізують завантажені в пам'ять об'єкти з базою даних. Для того, щоб це було можливим, після створення SQL-запиту, що перетворює об'єкт в SQL, отримані дані копіюються в поля об'єкта, як у всіх інших реалізаціях ORM. Після цього об'єкт повинен стежити за змінами цих значень і записувати їх у базу даних.

Системи управління реляційними базами даних показують хорошу продуктивність на глобальних запитах, які зачіпають велику ділянку бази даних, але об'єктно-орієнтований доступ ефективніший при роботі з малими об'ємами даних, оскільки це дозволяє скоротити семантичний провал між об'єктною і реляційною формами даних.

При одночасному існуванні цих двох різних світів збільшується складність об'єктного коду для роботи з реляційними базами даних, і він стає схильнішим до помилок. Розробники програмного забезпечення, що ґрунтується на базах даних, шукали легший спосіб досягнення постійності їхніх об'єктів.

Рішення

Розроблена безліч пакетів, що знімають необхідність в перетворенні об'єктів для зберігання в реляційних базах даних.

Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворять запити з одного вигляду в іншій. В результаті запиту об'єкта «чоловік» (з прикладу з адресною книгою) необхідний SQL-запит буде сформований і виконаний, а результати «магічним» чином перетворені в об'єкти «номер телефону» всередині програми.

З погляду програміста система повинна виглядати як постійне сховище об'єктів. Він може просто створювати об'єкти і працювати з ними як завжди, а вони автоматично зберігатимуться в реляційній базі даних.

На практиці все не так просто і очевидно. Всі системи ORM зазвичай проявляють себе в тому або іншому вигляді, зменшуючи в деякому роді можливість ігнорування бази даних. Більш того, шар транзакцій може бути повільним і неефективним (особливо в термінах згенерованого SQL). Все це може привести до того, що програми працюватимуть повільніше і використовувати більше пам'яті, чим програми, написані «вручну».

Але ORM позбавляє програміста від написання великої кількості коду, часто одноманітного і схильного до помилок, тим самим значно підвищуючи швидкість розробки. Крім того, більшість сучасних реалізацій ORM дозволяють програмістові при необхідності жорстко задати код SQL-запитів, який використовуватиметься при тих або інших діях (збереження в базу даних, завантаження, пошук тощо) з постійним об'єктом.

**22.Асоціативна модель даних**

*Асоціативна модель даних* - модель представлення даних, в якій база даних складається з двох типів структур даних - елементів і посилань, що зберігаються в єдиній однорідної загальній структурі в якості альтернативи реляційної та об'єктної моделям даних. Близька до моделі даних cущность-зв'язок.  
Крім загального опису моделі, існує реалізація і набір методів, запатентовані Уїльямсом.  
Сутність моделі: Реальний світ моделюється компонентами двох сортів: *сутностями* (елементами) і *асоціаціями* (зв'язками). *Сутність* - це щось, що існує окремо і самостійно. *Асоціація* - це щось, що існує тільки в зв'язку з якимись іншими речами, і яке не має сенсу у відриві від них.  
В рамках бази даних сутності й асоціації є структурами даних, що володіють такими властивостями:**1)** "Сутності" мають унікальний ідентифікатор, ім'я та тип.  
**2)** "Асоціації" мають унікальний ідентифікатор, а також містять ідентифікатори трьох інших об'єктів, що представляють "джерело", "дієслово" і "мета", які описують якийсь факт про "джерело". Кожен з цих трьох об'єктів може бути як "сутністю", так і "асоціацією".  
Наприклад, наступне твердження: "Рейс SU1234 прибуває в аеропорт" Пулково "12 серпня 2010 року в 13:40" в рамках цієї моделі містить сім "сутностей":  
♣ чотири "іменників": "Рейс SU1234", "аеропорт" Пулково "", "12 серпня 2010 року" і "13:40";  
♣ і три "дієслова": "прибуває в", "[на дату]", "в".  
Взаємозв'язки між "сутностями" описуються наступними асоціаціями:  
♣ <X>: "Рейс SU1234" "прибуває в" "аеропорт" Пулково ""  
♣ <Y>: <X> "[на дату]" "12 серпня 2010 року"  
♣ <Z>: <Y> "в" "13:40"

23. **Модель даних типу Сутність-Атрибут-Значення (EAV)**

EAV являє собою універсальну структуру опису даних, в якій дані зберігаються фактично в одному

лінійному списку. У EAV кількість критеріїв, які можуть бути використані, є потенційно великою, а кількість

таблиць у базі даних при цьому не змінюється. Розглянемо складові моделі:

1) сутність − абстрактний елемент моделі, який не має чітко виражених характеристик і спроможний

додавати до себе будь-які критерії;

2) атрибут − це конкретний критерій (характеристика), що додається до сутності;

3) значення − це елемент, який відображає можливі значення конкретного критерію (характеристики).

Для кращого розуміння підходу наведемо приклад: мобільний телефон (сутність), об’єм пам’яті

(атрибут), 1 Gb (значення). З цього можна зробити висновок, що при великих кількостях та частих

коригуваннях критеріїв найкраще підходить модель даних EAV, адже незважаючи на потужні запити і велику

кількість індексів, ця модель дає дуже велику гнучкість і простоту використання. Але варто зазначити, що EAV

− це лише модель даних і продуктивність її залежить від правильності реалізації бізнес-логіки.

**24. Контекстна модель**   
Модель контекст даних можна розглядати як набір об'єктно-орієнтованих, мережевих і слабоструктурованих моделей. Іншими словами, це гнучка модель, ви можете використовувати будь-який тип структури бази даних в залежності від завдання.

Така модель даних реалізована в СУБД ConteXt .  
КЛАС основний модуль зберігання інформації в ConteXt. Клас містить МЕТОДИ і описує Об'єкт. Об'єкт містить ПОЛЯ і ВЛАСТИВОСТІ.  
Поле може бути складовим і може містити інші поля. Властивістю є безліч полів, що відносяться до конкретного об'єкта.

Контекстна модель даних являє собою набір визначених типів і типів, визначених користувачем.  
Визначені типи включають в себе не тільки рядки символів, числа і тексти, але також посилання і агрегатні типи.  
Контекстна модель складається з трьох основних типів даних: звичайний,віртуальний і вказівниковий. Звичайне поле може бути атомарне або складове. Атомарне поле не має внутрішньої структури на відміну від складового, які можуть мати складну структуру і їх тип описується в заголовку класу.  
Складові поля діляться на статичні і динамічні.  
Інформація про тип статичного складеного поля зберігається в заголовку і не змінюється. Опис типу динамічного складеного інформація зберігається всередині об'єкта і може змінюватися від об'єкта до об'єкта.  
Подібно мережевим баз даних, крім полів безпосередньо зберігають інформацію, в контекстних БД є поля, що дозволяють дізнатися, де вона може бути знайдена, це ПОКАЖЧИКИ (або посилання) на об'єкти того або іншого класу.  
Оскільки основним модулем контекстної БД є об'єкт, то покажчик може посилатися тільки на об'єкт, а не на його поля. Покажчики також бувають двох видів: статичні і ДИНАМІЧНІ.  
На відміну від об'єктно-орієнтованих баз даних, контекстні бази даних не так тісно пов'язані з мовами програмування і не підтримують безпосередньо виклики з методів. Замість цього, виклик методів підтримується за допомогою концепції ВІРТУАЛЬНИХ полів.

Контекстну базу даних, яка не має складових полів, полів-покажчиків і властивостей можна вважати реляційною. З статичними складовими полями і полями-покажчиками, контекстна база даних стає об'єктно-орієнтованою. Якщо ж вона має тільки властивості, то в даному випадку є базою даних EAV. З динамічними складовими полями, її можна назвати частково-структурованою. Якщо в базі даних є всі перераховані типи, то тоді вона стає контекстної.

**25. NoSQL (not only SQL)**

*Ідея. Класифікація. Особливості архітектури. CAP-теорема.*

NoSQL (not only SQL) - термін, що позначає ряд підходів, проектів, спрямованих на реалізацію моделей баз даних, що мають суттєві відмінності від використовуваних в традиційних реляційних СУБД з доступом до даних засобами мови SQL. Опис схеми даних у разі використання NoSQL-рішень може здійснюватися через використання різних структур даних: хеш-таблиць, дерев та інших.

*Ідея:*1)Нереляційні модель даних 2)Відкритий вихідний код 3)Хороша горизонтальна масштабованість.

В якості одного з методологічних обгрунтувань підходу NoSQLвикористовується евристичний принцип, відомий як *теорема CAP,* який стверджує, що в розподіленій системі неможливо одночасно забезпечити узгодженість даних, доступність (англ. availability, в сенсі наявності відгуку з будь-якого запиту) і стійкість до розщеплення розподіленої системи на ізольовані частини. Таким чином, при необхідності досягнення високої доступності та стійкості до поділу передбачається не фокусуватися на засобах забезпечення узгодженості даних, що забезпечуються традиційними SQL-орієнтованими СУБД з транзакційними механізмами на принципах ACID

За своєю суттю *архітектура рішень NoSQL* орієнтована на боротьбу або з великим обсягом даних, або з їх підвищеною складністю.  
Відмінні риси NoSQL - нереляційні моделі даних, прості API або протоколи доступу, здатність до горизонтального масштабування на вимогу для деякого набору операцій на багатьох серверах, розподілене зберігання даних, ефективне використання розподілених індексів і пам'яті для запитів, досить вільне поводження з такими непорушними для традиційних СУБД речами, як транзакційна цілісність. Спільним для NoSQL-проектів є компроміси по відношенню до взаємно суперечливим вимогам - наприклад, відхід від підтримки стандартних правил для забезпечення транзакційної цілісності ACID (atomicity, consistency, isolation, durability, атомарность, узгодженість, ізольованість, довговічність) на користь горизонтальної масштабованості.Відмова від подібних вимог був неможливий і є догмою для традиційних СУБД. Дійсно, не можна бути одночасно надійним, швидким, розподіленим і цілісним, проте в ряді конкретних випадків можливі варіанти.

**26.Розподілені бази даних.**

*Фундаментальний принцип побудови. Наслідки. Типи архітектури.*

Різновидом баз даних з погляду їх формування, зберігання й використання є розподілені бази даних. Ці бази даних широко використовуються в організації комплексів взаємопов’язаних АРМ менеджерів, де встановлено ПЕОМ, а також у системі об’єктивного інформаційного забезпечення менеджменту.  
Розподілена база даних — це сукупність логічно пов’язаних баз даних або частин однієї бази, які розпаралелені між кількома територіально розподіленими ПЕОМ і забезпечені відповідними можливостями для управління цими базами чи їхніми частинами. Тож розподілена база даних реалізується на різних просторово розосереджених обчислювальних засобах разом з організаційними, технічними і програмними засобами її створення та ведення.  
До основних переваг розподіленої бази даних можна віднести:  
підвищення продуктивності системи за рахунок розпаралелення процесів збирання та обробки даних;підвищення ефективності управління базами даних і поліпшення експлуатаційних характеристик системи управління даними;досягнення збалансованості навантаження й синхронізації процесів збирання й обробки даних; підвищення надійності й живучості системи; вдосконалення гнучкості, нарощуваності та модифікованості бази даних;скорочення вартості організації і затрат на експлуатацію бази даних;збільшення обсягу збережених і доступних для обробки даних;  
зменшення обсягів даних, що пересилаються.  
Розподілені бази даних можна ефективно використовувати в предметних областях, які характеризуються: надто великими обсягами даних, що збираються, зберігаються й обробляються;  
фізичною розосередженістю місць збирання, зберігання й використання даних; наявністю розвинених засобів обчислювальної техніки та мереж передачі даних; можливістю збирання й обробки більшої частини інформації в місцях, де вона виникає чи зберігається; необхідністю одночасного масового збирання й обробки інформації тощо.   
За способом розміщення розподілені бази даних поділяють на зосереджені і розосереджені.  
Зосереджені (або централізовані) розподілені бази даних фізично розміщені в одному місці. Для обміну інформацією між окремими (локальними) підбазами використовуються канали зв’язку прямого доступу. Термін «канал зв’язку» (channel) щодо обміну інформацією означає засіб передавання інформації (письмової, усної, формальної, неформальної тощо), придатної для електронних засобів зв’язку..  
Розосереджені (або децентралізовані) розподілені бази даних фізично розміщені в різних місцях — вузлах обчислювальної мережі. Обмін інформацією між підбазами здійснюється з використанням каналів зв’язку. Як підбази розподіленої бази даних можуть використовуватися зосереджені (централізовані) бази даних і окремі (локальні) підбази. Обмін між взаємозв’язаними підбазами здійснюється здебільшого результатною (обробленою, узагальненою) інформацією. При виконанні запиту в таких системах використовується декомпозиція запиту на підзапит до локальних підбаз і паралельне виконання виділених підзапитів у різних вузлах обчислювальної мережі.   
Розглянута концепція розподілених баз даних набула значного поширення при функціонуванні розподіленої обробки інформації. При цьому організація такої обробки інформації відбувається, як правило, в рамках системи автоматизованого збирання й обробки інформації як в окремих структурних ланках, так і по об’єкту управління в цілому.

**27. Розподілені транзакції.**

*Протокол двофазової фіксації.*

Одним з дійсно чудових властивостей СУБД Oracle є її здатність прозоро обробляти розподілені транзакції. Я можу оновити дані в безлічі різних баз даних за одну транзакцію. Коли я виконую її фіксацію, то або фіксуються всі оновлення в усіх примірниках, або не фіксується ні одне з них (всі вони відкочуються). Мені не потрібен для цього ніякої додатковий код; я просто пишу commit;.

Ключем до розподілених транзакцій в Oracle є зв'язок баз даних (database link). Цей зв'язок є об'єкт бази даних, що описує спосіб реєстрації в іншому екземплярі бази з вашого екземпляра. Проте мета цього розділу не в тому, щоб розкрити синтаксис команд зв'язку баз даних (він повністю документований), а швидше просто в тому, щоб повідомити вас про його існування. Як тільки ви налаштуєте зв'язок баз даних, доступ до віддалених об'єктів стає дуже простим:

select \* from T @ another\_database;

Це дозволить вибрати дані з таблиці T в екземплярі бази даних, визначеному зв'язком ANOTHER\_DATABASE. Зазвичай ви повинні "приховувати" той факт, що T-дистанційна таблиця, створюючи її подання або синонім. Наприклад, можна виконати наступну команду і потім звертатися до T, як якщо б це була локальна таблиця:

create synonym T for T @ another\_database;

Тепер, маючи налаштовану зв'язок з віддаленою базою і отримавши можливість читати деякі таблиці, я також можу модифікувати їх (звичайно, за умови володіння відповідними привілеями). Виконання розподіленої транзакції тепер нічим не відрізняється від транзакції локальної. Ось все, що для цього потрібно:

update local\_table set x = 5;

update remote\_table @ another\_database set y = 10;

commit;

От і все. СУБД Oracle виконає фіксацію або в обох базах даних, або ні в одній з них. Вона використовує протокол 2PC (двофазної фіксації). Цей протокол дозволяє виконувати модифікації, що зачіпають безліч різних баз даних, фіксуючи їх автоматично. Він, наскільки можливо, намагається перекрити всі шляхи для розподілених збоїв перед тим, як виконати фіксацію. В 2PC між багатьма базами даних одна з баз - зазвичай та, до якої клієнт підключився спочатку - служить координатором розподіленої транзакції. Цей сайт запросить у яких сайтів готовності до фіксації. Тобто цей сайт звернеться до інших сайтів і попросить їх підготуватися до фіксації. Кожен з цих сайтів рапортує про своє "стані готовності", як "ТАК" або "НІ". Якщо будь-який з сайтів каже "НІ", виконується відкат всій транзакції. Якщо ж всі сайти рапортують "ТАК", сайт-координатор розсилає повідомлення з командою на виконання фіксації на всіх сайтах.

Це обмежує можливості для серйозних помилок. Перш ніж виконається "опитування" по 2PC, будь-яка розподілена помилка призведе до виконання відкоту на всіх сайтах. Не буде ніяких сумнівів щодо результату розподіленої транзакції. Після команди на фіксацію або відкат знову-таки немає ніяких сумнівів щодо результату розподіленої транзакції. Лише протягом дуже короткого періоду ("вікна") часу, коли координатор збирає відповіді, результат може бути неоднозначним після збою.

Припустимо, наприклад, що у нас є три сайти, що беруть участь в транзакції. Сайт 1 виступає в ролі координатора. Сайт 1 просить сайт 2 підготуватися до фіксації, і сайт 2 виконує це. Потім сайт 1 просити сайт 3 підготуватися до фіксації, і він також робить це. В цей момент часу сайт 1 - єдиний, хто знає про результат транзакції, і тепер він відповідає за сповіщення про це інших сайтів. Якщо помилка станеться прямо зараз - відбудеться збій мережі, сайт 1 залишиться без живлення або ще щось - сайти 2 і 3 залишаться в "підвішеному" стані. Вони отримають те, що називається сумнівної розподіленої транзакцією. Протокол 2PC намагається закрити "вікно" помилок, наскільки це можливо, але не може виключити таку вірогідність повністю. Сайти 2 і 3 повинні утримувати транзакцію відкритою, очікуючи нотифікації від сайту 1 команди на фіксацію або відкат. Якщо ви пам'ятаєте дискусію про архітектуру з глави 5, там було сказано,

що для вирішення цієї проблеми існує процес RECO. Це також той випадок, коли на сцену виходять COMMIT і ROLLBACK з опцією FORCE. Якщо причиною проблеми був збій мережі між сайтами 1, 2 і 3, то адміністратори сайтів 2 і 3 повинні звернутися до адміністратора сайту 1, запросити його про результат транзакції і виконати, відповідно, фіксацію або відкат вручну. Існує кілька (небагато) обмежень щодо того, що ви можете робити в розподіленої транзакції, і всі вони виправдані (на мій погляд, вони виглядають виправданими в будь-якому випадку). Нижче перераховані найбільш суттєві з них.

Ви не можете видати COMMIT по зв'язку баз даних. Тобто, не можна дати команду COMMIT @ удаленний\_сайт. Ви можете зафіксувати транзакцію тільки на сайті, який її ініціював.

Ви не можете виконувати DDL у віддаленій базі по зв'язку баз даних. Це - прямий наслідок попереднього обмеження. DDL виконує фіксацію. Не можна виконати фіксацію ні з одного сайту, крім ініціювання, а тому не можна виконувати DDL по зв'язку баз даних.

Не можна видати SAVEPOINT по зв'язку баз даних. Коротше кажучи, ви не можете видати жодного оператора управління транзакціями по зв'язку баз даних. Все управління транзакціями успадковується від сеансу, який спочатку відкриває зв'язку баз даних. Ви не можете здійснювати іншого управління транзакціями на розподілених примірниках в транзакції.

Недолік управління транзакціями по зв'язку баз даних також виправданий, оскільки ініціює сайт - єдиний, що має список усіх, хто залучений в транзакцію. Якщо в нашій конфігурації з трьох сайтів сайт 2 спробує зафіксувати транзакцію, у нього не буде жодних відомостей про те, що в ній бере участь сайт 3. Тому в Oracle тільки сайт 1 може видати команду на фіксацію. У цій точці для сайту 1 допустимо делегувати відповідальність за управління розподіленої транзакцією іншому сайту.

Ми можемо вказати, який сайт буде дійсним фіксуючим сайтом, встановлюючи параметр COMMIT\_POINT\_STRENGTH ("сила точки фіксації") сайту. Сила точки фіксації асоціює відносний рівень важливості з сервером в розподіленої транзакції. Чим важливіший сервер (більше доступних даних має бути на ньому), тим більш імовірно, що саме він буде координувати розподілену транзакцію. Вам може знадобитися це у випадку, коли необхідно виконати розподілену транзакцію між вашим робочим сервером і тестовим сервером. Оскільки координатор транзакції ніколи не сумнівається в результаті транзакції, буде краще, якщо робочий сервер стане координувати розподілену транзакцію. Вам не потрібно особливо турбуватися про ваш тестовому сервері - що на ньому раптом виявляться відкриті транзакції або блоковані ресурси. Вам безперечно варто турбуватися, якщо подібне трапиться на робочому сервері.

У неможливості виконання DDL по зв'язку баз даних взагалі немає нічого поганого. По-перше, DDL трапляється рідко. Ви виконуєте оператори DDL при інсталяції або оновлення. Робочі системи не виконують DDL (по крайней мере, не повинні). По-друге, все-таки існує спосіб запустити DDL через зв'язок баз даних - скористатися засобом планування завдань DBMS\_JOB або, в

Oracle 10g - пакетом планувальника DBMS\_SCHEDULER. Замість того щоб намагатися виконати DDL по зв'язку, ви використовуєте зв'язок для планування віддаленого завдання, щоб воно виповнилося, як тільки ви здійсните фіксацію. Таким чином, завдання запуститься на віддаленій машині, воно не буде частиною розподіленої транзакції і тому зможе виконати оператори DDL. Фактично це метод, за допомогою якого сервер реплікації Oracle (Oracle Replication Server) виконує розподілені команди DDL, щоб здійснити реплікацію схеми

**28. Інтеграція даних.**

*Способи інтеграції даних (Обєднані бази даних, сховища даних 889, посередники). OLAP. Вирішення проблем OLAP.*

Необхідність інтеграції даних виникає через неоднорідність програмного середовища, розподілений характер організації, підвищені вимоги до безпеки даних, необхідність наявності багаторівневих довідників метаданих, потребу в ефективному зберіганні й опрацюванні дуже великих обсягів інформації.Інтеґрація даних – це об'єднання даних, які спочатку вводяться в різні системи. Самі ці системи можуть розташовуватися в одній локальній мережі, але мати різні платформи і внутрішню архітектуру. Метою інтеґрації даних є отримання єдиної і цілісної картини корпоративних даних предметної області. Інтеґрація даних може бути описана за допомогою моделі, яка включає застосування, продукти, технології та методи.Існує три основні методи інтеґрації даних: консолідація, федералізація і розповсюдження.Консолідація даних – це збирання даних з територіально віддалених або різноплатформенних джерел даних в єдине сховище даних з метою їх подальшого опрацювання та аналізу.У середовищі сховищ даних однією з найпоширеніших технологій підтримки консолідації є технологія ETL (витягання, перетворення і завантаження – extract, transform, and load). Ще одна поширена технологія консолідації даних ECM - керування змістом корпорації (enterprise content management). Більшість рішень ECM напрямлені на консолідацію і керування неструктурованими даними, такими як документи, звіти і web-сторінки.Федералізація даних забезпечує єдину віртуальну картину одного або декількох первинних файлів даних. Процес федералізації даних завжди полягає у витяганні даних з первинних систем на підставі зовнішніх вимог. Всі необхідні перетворення даних здійснюються при їх витяганні з первинних файлів. Прикладом федералізації є інтеґрація корпоративної інформації (EII).Застосування розповсюдження даних здійснюють копіювання даних з одного місця в інше. Ці застосування зазвичай працюють в оперативному режимі і здійснюють переміщення даних до місць призначення, тобто залежать від певних подій. Оновлення в первинній системі можуть передаватися в кінцеву систему синхронно або асихронно. Прикладами технологій, що підтримують розповсюдження даних, є інтеґрація корпоративних застосувань (Enterprise application integration, EAI) і тиражування корпоративних даних (Еnterprise data replication, EDR). Методи, що використовуються застосуваннями інтеґрації даних, залежать як від потреб бізнесу, так і від технологічних вимог. Достатньо часто застосування інтеґрації даних використовує так званий гібридний підхід, який включає декілька методів інтеґрації. Приклад такого підходу – інтеґрація даних про клієнтів (customer data integration, CDI), метою якої є забезпечення узгодженої картини інформації про клієнтів.

OLAP (англ. online analytical processing, аналітична обробка в реальному часі) — це технологія обробки інформації, що дозволяє швидко отримувати відповіді на багатовимірні аналітичні запити. OLAP є частиною такого ширшого поняття, як бізнес-аналітика, що також включає такі дисципліни як реляційна звітність та добування данних (спосіб аналізу інформації в базі даних з метою відшукання аномалій та трендів без з'ясування смислового значення записів). Служить для підготовки бізнес-звітів з продажів, маркетингу, в цілях управління, для прогнозування, фінансової звітності та в схожих областях. Бази даних, сконфігуровані для OLAP, використовують багатовимірні моделі даних, що дозволяє виконувати складні аналітичні та спеціалізовані запити за коротокий проміжок часу. Вони запозичують окремі аспекти навігаційних та ієрархічних баз даних, які є швидшими за реляційні БД. Зазвичай результати OLAP-запитів представляють у формі матриць, де виміри складають рядки та колонки, а значеннями матриці є розміри. Головна причина використання OLAP для обробки запитів — це швидкість. Реляційні БД зберігають сутності в окремих таблицях, які зазвичай добре нормалізовані. Ця структура зручна для операційних БД (системи OLTP), але складні багатотабличні запити в ній виконуються відносно повільно. Зручнішою моделлю для виконання запитів (але не для внесення змін) є просторова БД. OLAP робить миттєвий знімок реляційної БД і структурує її в просторову модель для запитів. Заявлений час обробки запитів в OLAP складає близько 0,1% від аналогічних запитів до реляційної БД.

Типи Традиційно OLAP-системи поділяють на такі види: OLAP з багатьма вимірюваннями (Multidimensional OLAP), MOLAP; реляційна OLAP (Relational OLAP), ROLAP; гібридна OLAP (Hybrid OLAP), HOLAP.

MOLAP це класична форма OLAP, так що її часто називають просто OLAP. Вона використовує підсумовуючу БД, спеціальний варіант процесора просторових БД і створює необхідну просторову схему даних зі збереженням як базових даних, так і агрегатів. ROLAP працює безпосередньо з реляційним сховищем, факти і таблиці з вимірюваннями зберігаються в реляційних таблицях, і для зберігання агрегатів створюються додаткові реляційні таблиці. HOLAP використовує реляційні таблиці для зберігання базових даних і багатовимірні таблиці для агрегатів. Особливим випадком ROLAP є ROLAP реального часу (Real-time ROLAP, або R-ROLAP). На відміну від ROLAP, в R-ROLAP для зберігання агрегатів не створюються додаткові реляційні таблиці, а агрегати розраховуються у момент запиту. При цьому багатовимірний запит до OLAP-системи автоматично перетвориться в SQL-запит до реляційних даних.

Кожен тип зберігання має певні переваги, хоча є розбіжності в їх оцінці у різних виробників. MOLAP краще всього підходить для невеликих наборів даних, він швидко розраховує агрегати і дає відповіді, але при цьому генеруються величезні обсяги даних. ROLAP оцінюється як більш масштабоване рішення, яке до того ж використовує найменший можливий простір. При цьому швидкість обробки значно знижується. HOLAP знаходиться між цими двома підходами, він досить добре масштабується і швидко обробляється. Архітектура R-ROLAP дозволяє проводити багатовимірний аналіз OLTP-даних в режимі реального часу.

Складність в застосуванні OLAP полягає в створенні запитів, виборі базових даних і розробці схеми, внаслідок чого більшість сучасних продуктів OLAP поставляються разом з величезною кількістю заздалегідь сконфігурованих запитів. Інша проблема полягає в базових даних. Вони повинні бути повними і несуперечливими.

**29. Інформаційні системи.**

*Класифікація. Компоненти. Бізнес-процеси.*

Інформацíйна систéма (англ. Information system) — сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів. Класифікація:

За ступенем автоматизації. В залежності від ступеня (рівня) автоматизації виділяють ручні, автоматизовані й автоматичні інформаційні системи. Ручні ІС характеризуються тим, що всі операції з переробки інформації виконуються людиною. Автоматизовані ІС частина функції (підсистем) керування або опрацювання даних здійснюється автоматично, а частина — людиною. Автоматичні ІС усі функції керування й опрацювання даних здійснюються технічними засобами без участі людини (наприклад, автоматичне керування технологічними процесами).

За сферою призначення Оскільки ІС утворюються для задоволення інформаційних потреб в межах конкретної предметної галузі, то кожна предметна галузь (в сфері призначення) відповідає свій тип ІС. Перераховувати всі ці типи немає змісту, оскільки кількість предметних галузей велика, але можна вказати наприклад такі типи ІС: Економічна ІС — інформаційна система призначена для виконання функцій управління на підприємстві; Медична ІС — інформаційна система призначена для використання в лікувальному або лікувально-профілактичному закладі; Географічна ІС — інформаційна система, забезпечуюча збір, збереження, обробку, доступ, відображення і розповсюдження даних; Адміністративні; Виробничі; Навчальні; Екологічні; Криміналістичні; Військові та інші.

За місцем діяльності ІС Класифікація інформаційних систем за місцем діяльності: наукові ІС — призначені для автоматизації діяльності науковців, аналізу статистичної інформації, керування експериментом. ІС автоматизованого керування — призначені для автоматизації праці інженерів-проектувальників і розроблювачів нової техніки (технології). Такі ІС допомагають здійснювати: розробку нових виробів і технологій їхнього виробництва; різноманітні інженерні розрахунки (визначення технічних параметрів виробів, видаткових норм — трудових, матеріальних і т. д.); створення графічної документації (креслень, схем, планувань); моделювання проектованих об'єктів; створення керуючих програм для верстатів із числовим програмним керуванням. ІС організаційного керування — призначені для автоматизації функції адміністративного (управлінського) персоналу. До цього класу відносяться ІС керування як промисловими (підприємства), так і непромисловими об'єктами (банки, біржа, страхові компанії, готелі і т. д.) і окремими офісами (офісні системи). ІС керування технологічними процесами — призначені для автоматизації різноманітних технологічних процесів (гнучкі виробничі процеси, металургія, енергетика тощо).

Інформаційна система, як система управління, тісно пов’язується, як з системами збереження та видачі інформації, так і з іншої — з системами, що забезпечують обмін інформацією в процесі управління. Вона охоплює сукупність засобів та методів, що дозволяють користувачу збирати, зберігати, передавати і обробляти відібрану інформацію. За функціональним призначенням В залежності від функціонального призначення можна виділити такі системи: Керувальні (АСКТП, АСКВ); Проектувальні (САП); Наукового пошуку (АСНД, експертні системи); Діагностичні, моделювальні; [[Систем підготовки прийняття рішення]] (СППР).

Практично всі розглянуті різновиди інформаційних систем незалежно від сфери їхнього застосування включають один і тойже набір компонентів: функціональні компоненти: функціональні підсистеми (модулі, бізнеси-додатки), функціональні задачі, моделі й алгоритми; компоненти системи опрацювання даних: інформаційне забезпечення, програмное забезпечення, технічне забезпечення, правове забезпечення, лінгвістичне забезпечення; організаційні компоненти (персонал): нова організаційна структура фірми, персонал (штати, посадові інструкції).

**30. Проектування інформаційних систем.**

*Концептуальна, логічна, фізична моделі. Методології розробки інформаційних систем.*

Проектування інформаційних систем, що включають бази даних, здійснюється на фізичному і логічному рівнях. На логічному рівні проектується концептуальна модель даних. Концептуальна (інфологічна) модель представляє об’єкти та взаємозв’язки між ними без визначення способів їхнього фізичного збереження.Поєднуючи уявлення про вміст бази даних, одержані в результаті опитування майбутніх користувачів, і свої власні уявлення про дані, що можуть знадобитися в майбутніх додатках, спочатку створюють інфологічну модель даних. Інфологічна модель даних - це опис предметної області, виконаний природною мовою, за допомогою математичних формул, графіків, таблиць тощо Концептуальна модель транслюється потім в модель даних, сумісну з обраною системою управління базами даних, іншими словами створюється датологічна модель. Датологічна модель даних - це опис предметної області, виконаний мовою обраної системи управління базами даних Версія концептуальної моделі, що може бути забезпеченою певною системою управління базами даних, називається логічною моделлю. Логічна модель відображає логічні зв’язки між елементами даних, незалежно від їхнього змісту та середовища збереження. Фізична модель визначає розміщення даних, методи доступу та техніку індексування. Рішення проблем проектування на фізичному рівні багато в чому залежить від системи управління базами даних, що використовується. Воно автоматизоване і сховане від користувача. Збережені в базі дані мають визначену логічну структуру, тобто представлені деякою моделлю, що підтримує система управління базами даних. До числа найважливіших відносяться такі моделі даних: ієрархічна; мережна; реляційна; об’єктно - орієнтована. В ієрархічній моделі дані представляються у вигляді деревоподібної (ієрархічної) структури. Вона зручна для роботи з ієрархічно упорядкованою інформацією і громіздка для інформації зі складними логічними зв'язками. Недоліком мережної моделі даних є висока складність і жорсткість схеми бази даних, побудованої на її основі. Реляційна модель даних (РМД) назву одержала від англійського терміна relation - відношення. Її запропонував у 70-х роках співробітник фірми IBM Эдгар Кодд. При дотриманні визначених умов відношення представляється у вигляді двохвимірної таблиці, звичної для людини. Більшість сучасних баз даних для персональних електронно-обчислювальних машин є реляційними. Об’єктно – орієнтовані бази даних поєднують у собі дві моделі даних, реляційну і мережну, що використовуються для створення баз даних зі складними структурами даних. Модель даних у загальному випадку описує набір базових ознак, які повинні мати всі конкретні системи управління базами даних і керовані ними бази даних, засновані на цій моделі.